Remarks

Claims 1, 4-5, 7-8 and 10 have each been amended the recite that the method in question comprises the steps of (1) optimizing the loop process by performing the previously recited steps and (2) transferring execution from the interpreter process to the optimized loop process via one of the transfer points. The remaining claims have been amended to conform with these changes.

Claim Rejections—35 U.S.C. § 112

1. 35 U.S.C. § 112, First Paragraph

Claims 1-10 stand rejected under 35 U.S.C. § 112, first paragraph, as failing to comply with the enablement requirement (pages 2-3, ¶ 4). The Examiner asserts that, "[a]s known to those of ordinary skill in the art," the loop process of a compiled code process is interpreted to refer "to the 'main loop' of the program which repeats the princip[al] function of the program until the program is terminated." The Examiner goes on to say that the originally filed application appears to deal with transfer points in a programmatic loop, and that the specification "is silent on moving transfer points to relative positions around the main loop of a compiled code process."

The Examiner offers no support for his assertion that the term "loop process" would be interpreted by those of ordinary skill in the art to refer to the main loop of a program, as distinguished from an inner loop. The specification uses the term "loop process" interchangeably with the term "loop",[1] which the online encyclopedia Wikipedia defines as "a sequence of statements which is specified once but which may be carried out several times in succession".[2] Nothing in this definition limits it to a main loop as argued

[1] Compare, for example, the language in the summary portion on page 5 (which refers to a "loop process") with the language in the detailed description on page 7 (which refers to a "loop").

[2] See article entitled "Control flow", found at http://en.wikipedia.org/wiki/Program_loop.

by the Examiner. Moreover, the same online encyclopedia separately defines "main loop" as follows (hyperlinks omitted):[3]

> In computing, the **main loop** (sometimes called the **event loop** or **main event loop**) is a common design approach, typically used by applications featuring an event driven graphical user interface. It consists of a loop which continuously executes, polling for user events and handling each one. After handling the event in whatever manner is appropriate, control returns to the loop until another event is received, and so on. The loop is at the highest level of control within the program, hence 'main'.

Clearly, then, when the art wished to refer to only a main loop, it would use that term and not the more general term "loop" (or "loop process").

Even if there were some support (which there is not) for the proposition that "loop process" is generally understood to refer to a main program loop, that is not the applicable standard. Rather, the standard is how the term is used in applicants' specification. As the Court of Appeals for the Federal Circuit has declared most recently in Phillips v. AWH Corp., 75 USPQ2d 1321 (Fed. Cir. 2005):

> [O]ur cases recognize that the specification may reveal a special definition given to a claim term by the patentee that differs from the meaning it would otherwise possess. In such cases, the inventor's lexicography governs. See CCS Fitness, Inc. v. Brunswick Corp., 288 F.3d 1359, 1366 [62 USPQ2d 1059] (Fed. Cir. 2002).

Id. at 1329.

[3] See article entitled "Main loop", found at http://en.wikipedia.org/wiki/Main_loop.

Thus, the terms "loop" and "loop process" as used in applicants' specification refer to any loop in the compiled code process, and not just a main program loop as suggested by the Examiner. Accordingly, the Examiner's rejection of claims 1-10 under 35 U.S.C. § 112, first paragraph, is untenable and should therefore be withdrawn.[4]

2. 35 U.S.C. § 112, Second Paragraph

Claims 4 and 8-10 also stand rejected under 35 U.S.C. § 112, second paragraph, as being "incomplete for omitting essential elements" (page 3, ¶ 6). Specifically, the Examiner alleges that while the preamble of each claim recites a program storage device, the remainder of each claim "does not provide supporting tangible elements which are necessary for such a program storage device to perform the invention" (Id.). The Examiner has suggested that each claim element be prefixed with something such as "computer executable instructions for moving . . ." or the like. Applicants respectfully disagree.

The format of these claims is modeled after one of the formats[5] involved in In re Beauregard, 35 USPQ2d 1383 (Fed. Cir. 1995), which found the format to be statutory. The format in that case was the following:

> 2. A program storage device readable by a machine, tangibly embodying a program of instructions executable by the machine to perform method steps for [overall function of software], said method steps comprising:
>
> [conventional method claim elements describing software].

This "program storage device" claim format is also found in numerous U.S. patents issued subsequently, including at least four patents having the same Primary Examiner as this case: U.S. Patents 6,678,745, 6,772,411, 6,817,013 and 6,883,163. In none of these patents are the method

[4] It is unclear how the assertion that programmatic loops are analyzed "in the context of an abstract flow graph" (in particular, a control flow graph) is supposed to advance the Examiner's argument. In any event, control flow graphs (CFGs) are a standard method of analyzing program flows, as documented in the online encyclopedia Wikipedia in the article entitled "Control flow graph" (http://en.wikipedia.org/wiki/Control_flow_graph).

steps prefixed by the "computer executable instructions" language suggested by the Examiner. Rather, the preamble recitation that the instructions are executable by the machine to perform the specified method steps sufficiently ties the program instructions to the method steps, in the issued patents as well as the instant application. Accordingly, claims 4 and 8-10 comply fully with 35 U.S.C. § 112, second paragraph.

Claim Rejections—35 U.S.C. § 101

Claims 1-10 stand rejected under 35 U.S.C. § 101 as being directed to non-statutory subject matter (page 4, ¶¶ 7-8). Allegedly, the claimed methods are not supported by any hardware-implemented elements that produce "tangible results" (Id.).

As noted above, applicants have amended each of the independent claims to recite that the method in question comprises the steps of (1) optimizing the loop process by performing the previously recited steps and (2) transferring execution from the interpreter process to the optimized loop process via one of the transfer points.[6] Since the execution of a loop process clearly implies a hardware element (the machine on which the process is executed) as well as "tangible results" (if only the indexing of an instruction address register), the claims as amended are undoubtedly statutory.

Claim Rejections—35 U.S.C. § 103

Finally, claims 1-10 stand rejected as being unpatentable over Aho et al., Compilers: Principles, Techniques and Tools, Chapter 10, "Code Optimization", pages 585-722 (1986) ("Aho") in view of Bak et al., U.S. Patent 6,513,156 ("Bak") and Bacon et al., "Compiler Transformations for High-Performance Computing", ACM Computing Surveys, vol. 26, no. 4, pages 345-420 (Dec. 1994) ("Bacon") (page 5, ¶ 11). This rejection is respectfully traversed.

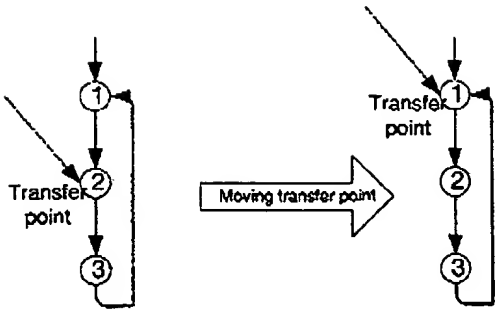
[5] This format is not found in the cited decision itself, but is reproduced in the resulting patent (U.S. Patent 5,710,578), as well as any of a number of articles commenting on the case.

[6] The transfer step is shown as step S18 in Fig. 2.

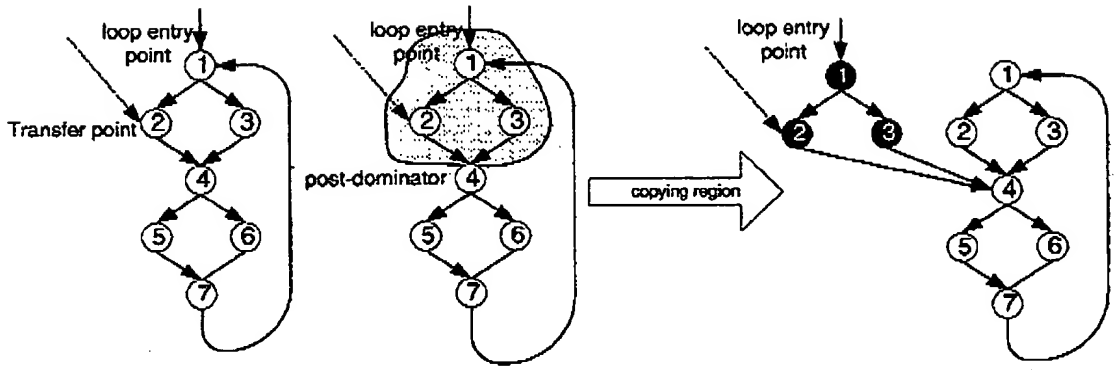
Each of the rejected claims recites either the movement of transfer points to the top of a loop process, the copying of code from the top of a loop process, or both. Since either of these recitations is sufficient to distinguish over the art cited, it will be sufficient for the purposes of this response to address these two aspects of applicants' invention.

1. Moving Transfer Points to Top of Loop Process

This aspect of applicants' claimed invention, recited in claims 1-6 and 8-9, involves moving one or more transfer points, at which program execution is transferred from an interpreter process to a loop process of a compiled code process, to the top of the loop process if they can be moved there without a problem occurring. This is shown in Fig. (a) below, an example also discussed in a previous amendment. Fig. (a) shows on the left a loop (nodes 1-3) having an entry point (node 1) at its top as well as a transfer point (node 2) along its length to which control may pass other than through the entry point (i.e., from another routine). Because of the transfer point at node 2, the loop is irreducible and many common optimization techniques cannot be used. In accordance with this aspect of the invention, the transfer point is moved (if this can be done without a problem occurring) from node 2 to node 1 at the top of the loop, as shown on the right in the figure, so that the loop is now reducible.



(a) An example of moving transfer point



(b) An example of copying a region from entry point to the post-dominating point

In addressing this aspect of applicants' claimed invention, the Examiner first notes Aho's various discussions of code motion (pages 596 and 638-642), as well as the discussion of "transfer points" (pages 653-660), all of which "would inform any optimizations involving loop invariant computations" (page 5, ¶ 11)). The Examiner concedes that Aho does not "expressly disclose" moving transfer points to the top of a loop, but contends that this step would be "obvious in any code that contains transfer points under analysis as a candidate for optimization as discussed on page 659: 'Use of Change Information', which discusses such analysis for the purposes of common subexpression identification" (page 5, ¶ 11). The Examiner concludes that one of ordinary skill in the art "would be motivated to determine the impact of a procedure on variables in a program in order to determine if the procedure could be optimized (page 655 paragraph 3)" (Id.). Applicants respectfully disagree.

The code motion discussed in the excerpts cited above deal with the movement of code to outside of a loop (in particular, before the loop) if the values generated are the same for any iteration of the loop. This aspect of applicants' invention, however, does not involve the movement of code; rather, it involves the movement of a transfer point to a specified position in a loop. So it is not seen how Aho's discussion of code motion is even relevant here or how it would "inform" a person of ordinary skill in the art.

As for the discussion on pages 653-660, including the subsection on pages 659-660 entitled "Use of Change Information", this is from a section entitled "Dealing with Aliases", which treats the problem of two or more expressions denoting the same memory address (page 648). Again, while this may be an interesting optimization problem, it is not seen how this would inform the situation of a loop having a transfer point that is separated from the top of the loop. Accordingly, nothing in any of these excerpts from Aho would suggest moving a transfer point in the manner claimed by applicants.

Finally on this subject, while the Examiner has not applied it as a reference, he cites Hirai et al. U.S. Patent 5,862,384 ("Hirai"), which he considers pertinent inasmuch as it "discloses moving a transfer point to the top of a loop in order to promote optimization" (page 10, ¶ 12). The Examiner refers in particular to Figs. 11-12, in which loop-invariant statements 1107 and 1110 are moved from inside a loop block 1105 delimited by statements 1106 and 1116 to a position just before the loop block (col. 23, lines 61-67).

While Hirai may be pertinent to loop optimization generally, it does not disclose moving a transfer point to the top of a loop as asserted by the Examiner. The statements 1106 and 1116 that are moved outside of the loop were never transfer points into the loop; indeed, the only entry point to the loop block 1105, both before and after optimization, is via statement 1106. Rather, Figs. 11-12 of Hirai are just another example of Aho's "code motion" technique described at page 596, in which loop-invariant expressions are moved outside of the loop.

Accordingly, neither Aho nor Hirai teaches moving one or more transfer points at which program execution is transferred to a loop process to the top of the loop process as claimed by applicants. Therefore, claims 1-6 and 8-9, which are directed to this aspect of applicants' invention, distinguish patentably over the art cited by the Examiner.

2. Copying Code from Top of Loop Process

This aspect of applicants' invention, recited in claims 1-4, 6-7 and 9-10, involves copying code, extending from the top of the loop process to a point that post-dominates the top of the loop process and the transfer points, to a location immediately preceding the loop process if the transfer points are located inside the loop process. This is shown in Fig. (b) above. At the very left of that figure is shown a loop (nodes 1-7) having an entry point (node 1) and a transfer point (node 2) to which control may pass other than through the entry point. Node 4 of the loop is said to post-dominate nodes 1 and 2 since every path of execution passing through node 1 or 2 subsequently passes through node 4. Again, because of the transfer point at node 2, the loop is irreducible and many common optimization techniques cannot be used.

In accordance with this aspect of the invention, the region (nodes 1-3) from the top of the loop (node 1) to a point (node 4) that post-dominates the top of the loop and the transfer point (node 2) is copied to a location immediately preceding the loop. More particularly, nodes 1-3 have been copied as nodes 1'-3' (as the nodes shown as black circles will be identified here), with node 1' having directed edges to nodes 2' and 3' and with nodes 2' and 3' having directed edges to post-dominating node 4. With this change, the loop containing nodes 1-7 has but a single entry point (now node 4), allowing conventional optimization techniques to be used.

The Examiner points to several excerpts from Aho, the only reference cited on this aspect of code copying. None of these excerpts, however, teach this aspect of applicants' claimed invention. The Examiner first points to the discussion of various code optimizations at pages 664-668, in particular the discussion of node splitting beginning at the bottom of page 666. More relevantly, the Examiner points to the node-splitting example shown in Fig. 10.49 on page 668. There, the node 2 of Fig. 10.49(a) is split into the pair of nodes 2a and 2b shown in Fig. 10.49(b),

permitting the further simplifications shown in Fig. 10.49(c) and (d). However, while this shows copying of a node, there are no depicted entry points or transfer points to the flow graph formed by the nodes illustrated in Fig. 10.49(a). Thus, this portion of Aho does not teach copying code represented by a set of (one or more) nodes, extending from the top of a loop to a point that post-dominates the top of the loop and a transfer point, to a location immediately preceding the loop as claimed by applicants.

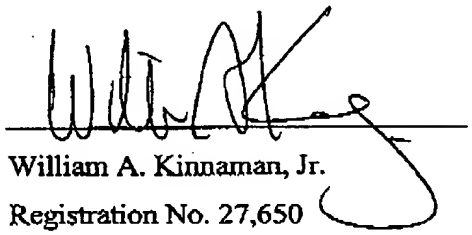
The same comments apply to the node splitting example shown in Fig. 10.57 on page 680 and described on pages 679-680. There, the node on the left, containing a pair of dots, is split into a pair of such nodes, in the same manner as node 2 in Fig. 10.49. Again, this only shows node splitting in the abstract, without any relation to entry points or transfer points to a loop. It does not teach copying code represented by a set of such nodes, extending from the top of a loop to a point that post-dominates the top of the loop and a transfer point.

Accordingly, Aho fails to teach copying code, extending from the top of a loop process to a point that post-dominates the top of the loop process and one or more transfer points, to a location immediately preceding the loop process as claimed by applicants. Therefore, claims 1-4, 6-7 and 9-10, which are directed to this aspect of applicants' invention, distinguish patentably over the art cited by the Examiner.

Conclusion

Entry of this amendment and reconsideration of the application as so amended are respectfully requested. It is hoped that upon such consideration, the Examiner will hold all claims allowable and pass the case to issue at an early date. Such action is earnestly solicited.

Respectfully submitted,
TOSHIAKI YASUE et al.

By 

William A. Kinnaman, Jr.
Registration No. 27,650
Phone: (845) 433-1175
Fax: (845) 432-9601

WAK/wak

